Quantum Programming Languages

Benoît Valiron (CentraleSupélec / LMF)

Nov 2025

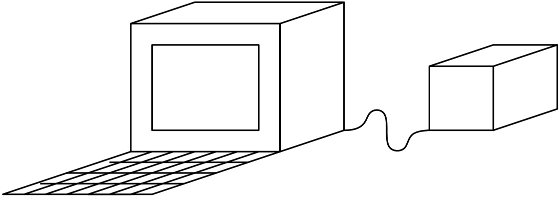
1^{st} QCOMICAL School

Plan

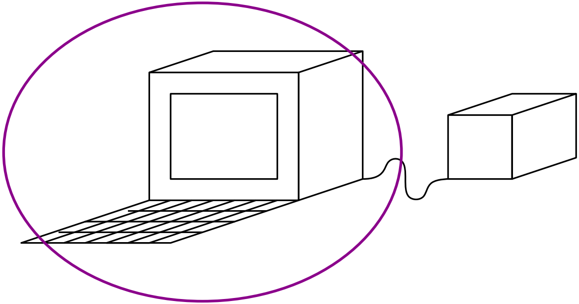
Plan

Plan

Model of Computation: Co-processor

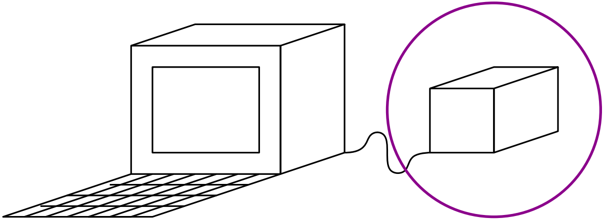


Model of Computation: Co-processor



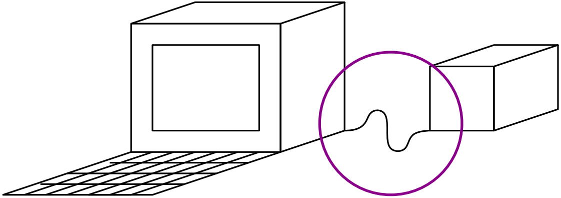
The program lives here

Model of Computation: Co-processor



This only holds the quantum memory

Model of Computation: Co-processor



Series of instructions/feedbacks

The Quantum Memory

A quantum memory

- Contains individually addressable quantum registers (qbits)
- State of n qbits: complex combination of strings of n bits
- E.g. for $n = 3$:

$$\begin{array}{rcl} & -\frac{1}{2} & \cdot\, 0\,0\,0 \\ + & \frac{1}{2} & \cdot\, 0\,0\,1 \\ + & \frac{i}{2} & \cdot\, 1\,1\,0 \\ - & \frac{i}{2} & \cdot\, 1\,1\,1 \end{array}$$

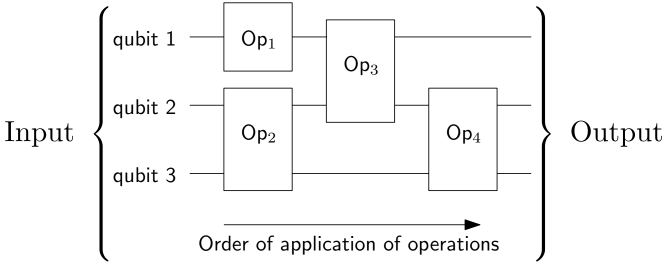
- With a norm condition.

Unlike probabilistic distributions,

all are available at the same time.

Quantum Circuit Model

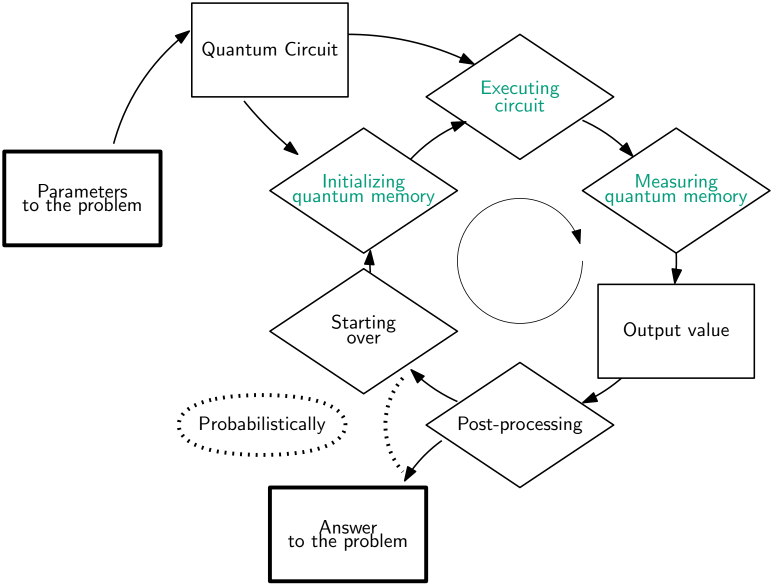
Stream of instructions: a series of elementary gates applied on the quantum memory, that are described by a quantum circuit.



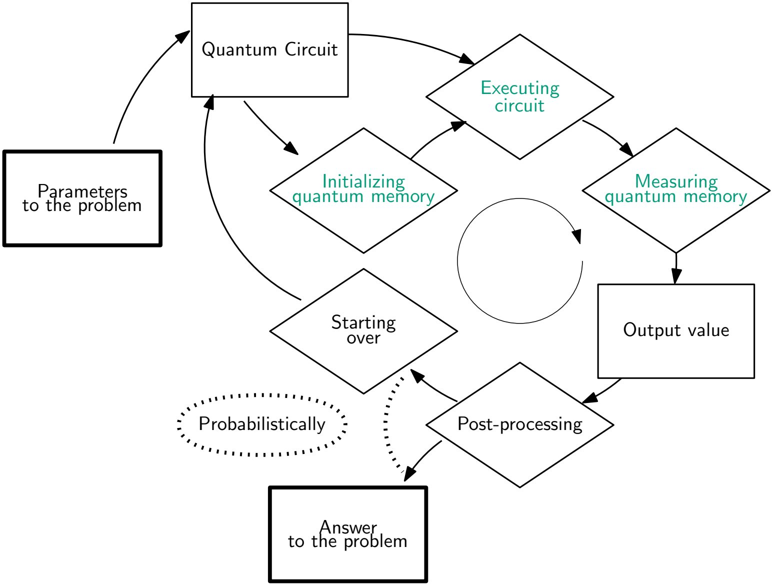
- Each operation is reversible, unitary on the space of states
- Wire $\equiv$ quantum bit $\equiv$ a quantum register
- No “quantum loop”, “conditional stop” nor “branching point”

Plan

Structure of (Static) Quantum Algorithms



Structure of (Variational) Quantum Algorithms



Quantum Algorithm, Probabilistic Algorithm

Simple probabilistic algorithm to factor 289884400687823

- Fair draw of a number among $2, 3, 4, 5, \ldots$
- Test: Euclidian division
- Found a factor: success. Otherwise: start over.

Very poor probability of success!

Shor's factorization algorithm

- Probabilistic sampling performed with measurement
- The quantum circuit build a "good" probability distribution.
  $\rightarrow$ boosts factors!

Quantum programming means building a circuit

(In case you're wondering: $315697 \cdot 918236159$)

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Quantum Fourier Transform
     Assuming $\omega = 0.\mathtt{xy}$, we want

$$\begin{array}{rl} & (e^{2\pi i\omega})^0 \cdot 00 \\ + & (e^{2\pi i\omega})^1 \cdot 01 \\ + & (e^{2\pi i\omega})^2 \cdot 10 \\ + & (e^{2\pi i\omega})^3 \cdot 11 \end{array} \qquad \longmapsto \qquad 1 \cdot \mathtt{xy}$$

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Phase estimation.
   - Amplitude amplification.
     Qubit 3 in state 1 means good.

$$\begin{array}{rl} & \alpha_0 \cdot 000 \\ + & \alpha_1 \cdot 011 \\ + & \alpha_2 \cdot 100 \\ + & \alpha_3 \cdot 110 \end{array} \longmapsto \begin{array}{rl} & {\scriptstyle \alpha_0} \cdot 000 \\ + & \Large{\alpha_1} \cdot 011 \\ + & {\scriptstyle \alpha_2} \cdot 100 \\ + & {\scriptstyle \alpha_3} \cdot 110 \end{array}$$

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Quantum Fourier Transform
   - Amplitude amplification.
   - Quantum walk.

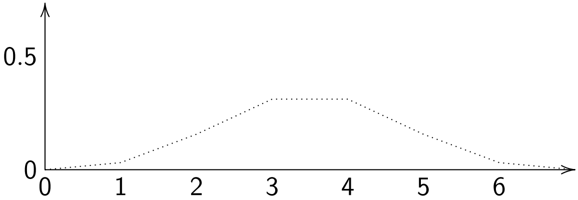
$$\begin{tikzcd}
0 \arrow[r] & 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \arrow[r] & 5 \arrow[r] & 6 \arrow[r] & 7 \arrow[d] \\
15 \arrow[u] & 14 \arrow[l] & 13 \arrow[l] & 12 \arrow[l] & 11 \arrow[l] & 10 \arrow[l] & 9 \arrow[l] & 8 \arrow[l]
\end{tikzcd}$$

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Quantum Fourier Transform
   - Amplitude amplification.
   - Quantum walk.

     After 5 steps of a probabilistic walk:

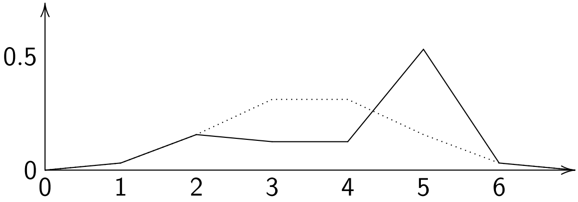


Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Quantum Fourier Transform
   - Amplitude amplification.
   - Quantum walk.

     After 5 steps of a quantum walk:



Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

1. Quantum primitives.
   - Quantum Fourier Transform
   - Amplitude amplification
   - Quantum walk
   - Hamiltonian simulation
   - ...

They are given as circuit templates

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

2. Oracles.
   - Take a classical function $f : \texttt{Bool}^n \to \texttt{Bool}^m$.
   - Construct

$$\begin{array}{rccc} \overline{f}: & \texttt{Bool}^{n+m} & \longrightarrow & \texttt{Bool}^{n+m} \\ & (x,y) & \longmapsto & (x, y \oplus f(x)) \end{array}$$

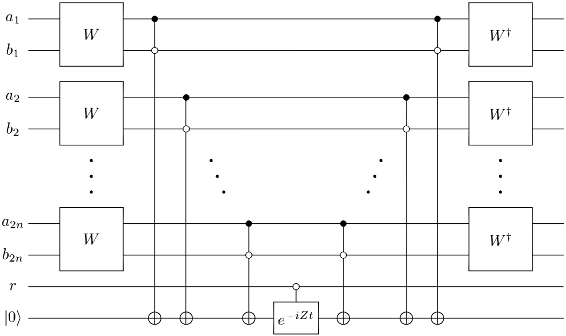
   - Build the unitary U_f acting on $n+m$ qubits computing $\overline{f}$.

Building the circuit depends on how f is given

Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

3. Blocks of loosely-defined low-level circuits.

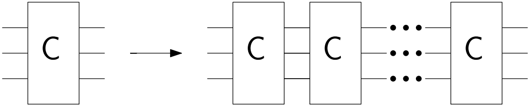


This is not a formal specification!

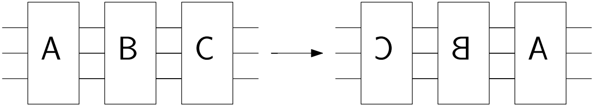
Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

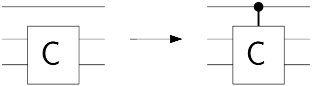
4. High-level operations on circuit:
   - Repetition



   - Inversion

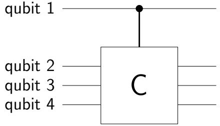


   - Control



The techniques used to described quantum algorithms are diverse.

4. High-level operations on circuit:
   - Control : conditional action of a circuit



C is applied on qubits 2-4 only when qubit 1 is true:
Suppose that C flips its input bits. Then the above circuit does

$$\begin{array}{rl} \texttt{qbit} & 1\,234 \\ \frac{1}{\sqrt{2}} & {\color{blue}1}\,010 \\ +\frac{1}{\sqrt{2}} & {\color{red}0}\,110 \end{array} \longmapsto \begin{array}{rl} \texttt{qbit} & 1\,234 \\ \frac{1}{\sqrt{2}} & {\color{blue}1\,101} \\ +\frac{1}{\sqrt{2}} & {\color{red}0}\,110 \end{array}$$

This acts as a form of "quantum test"

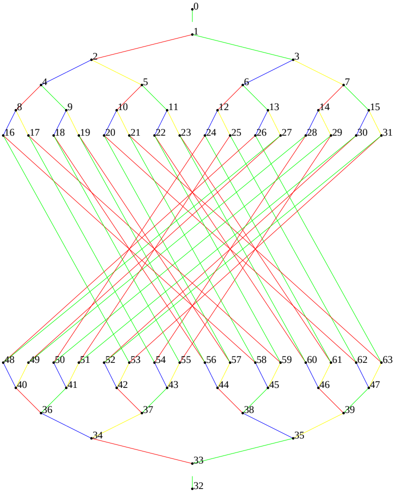
Internals of Current Quantum Algorithms

The techniques used to described quantum algorithms are diverse.

5. Classical processing.
   - Generating the circuit…
   - Computing the input to the circuit.
   - Processing classical feedback in the middle of the computation.
   - Analyzing the final answer (and possibly starting over).

Plan

Case study: BWT algorithm



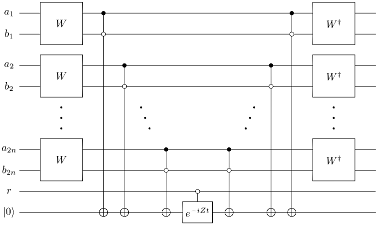
- Start at entrance, look for exit
- Description of the graph:

$$\begin{array}{lcl} I & : & \texttt{Node} \\ G & : & \texttt{Color} \times \texttt{Node} \to \texttt{Maybe Node} \\ O & : & \texttt{Node} \to \texttt{Bool} \end{array}$$

- Random/Quantum walk
- Parameters:
  height of tree ; number of steps.

Case study: BWT algorithm

- Initialization of a register to the input node (using I)
- 10^6 iterations:
  - Diffuse
  - Call oracle for red
  - Diffuse
  - Call oracle for green
  - Diffuse
  - Call oracle for blue
  - Diffuse
  - Call oracle for yellow
- Measure the node we sit on
- Test with O that we reached the output node.



Case study: QLS algorithm

Considering a vector $\vec{b}$ and the system

$$A \cdot \vec{x} = \vec{b},$$

compute the value of $\quad \langle \vec{x} \,|\, \vec{r} \rangle \quad$ for some vector $\vec{r}$.

Practical situation: the matrix A corresponds to the finite-element approximation of the scattering problem:

Case study: QLS algorithm

For more precision: `arXiv:1505.06552`

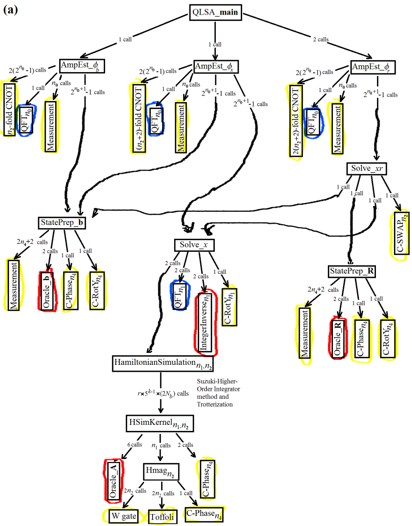
Three oracles:

- for $\vec{r}$ and for $\vec{b}$: input an index, output (the representation of) a complex number
- for A: input two indexes, output also a complex number

It uses many quantum primitives

- Amplitude estimation
- Phase estimation
- Amplitude amplification
- Hamiltonian simulation

Case study: QLS algorithm



» Yellow: Elementary gates.

» Red: Oracles.

» Blue: QFT's.

» Black: Subroutines.

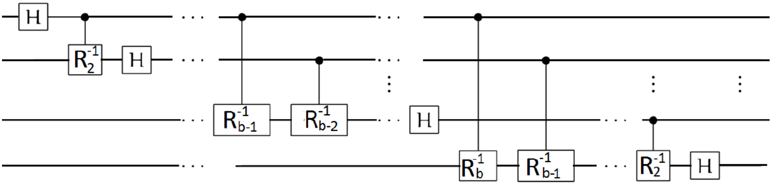
» *Parameters:*
Dimensions of the space;
Precision for each of the vectors;
Allowed error;
Various parameters for A...
In total, 19 parameters.

Case study: QLS algorithm

Oracle R is given by the function

```
calcRweights y nx ny lx ly k theta phi =
    let (xc',yc') = edgetoxy y nx ny in
    let xc = (xc'-1.0)*lx - ((fromIntegral nx)-1.0)*lx/2.0 in
    let yc = (yc'-1.0)*ly - ((fromIntegral ny)-1.0)*ly/2.0 in
    let (xg,yg) = itoxy y nx ny in
    if (xg == nx) then
        let i = (mkPolar ly (k*xc*(cos phi)))*(mkPolar 1.0 (k*yc*(sin phi)))*
                ((sinc (k*ly*(sin phi)/2.0)) :+ 0.0) in
        let r = ( cos(phi) :+ k*lx )*((cos (theta - phi))/lx :+ 0.0) in i * r
    else if (xg==2*nx-1) then
        let i = (mkPolar ly (k*xc*cos(phi)))*(mkPolar 1.0 (k*yc*sin(phi)))*
                ((sinc (k*ly*sin(phi)/2.0)) :+ 0.0) in
        let r = ( cos(phi) :+ (- k*lx))*((cos (theta - phi))/lx :+ 0.0) in i * r
    else if ( (yg==1) && (xg<nx) ) then
        let i = (mkPolar lx (k*yc*sin(phi)))*(mkPolar 1.0 (k*xc*cos(phi)))*
                ((sinc (k*lx*(cos phi)/2.0)) :+ 0.0) in
        let r = ( (- sin phi) :+ k*ly )*((cos(theta - phi))/ly :+ 0.0) in i * r
    else if ( (yg==ny) && (xg<nx) ) then
        let i = (mkPolar lx (k*yc*sin(phi)))*(mkPolar 1.0 (k*xc*cos(phi)))*
                ((sinc (k*lx*(cos phi)/2.0)) :+ 0.0) in
        let r = ( (- sin phi) :+ (- k*ly) )*((cos(theta - phi)/ly) :+ 0.0) in i * r
    else 0.0 :+ 0.0
```

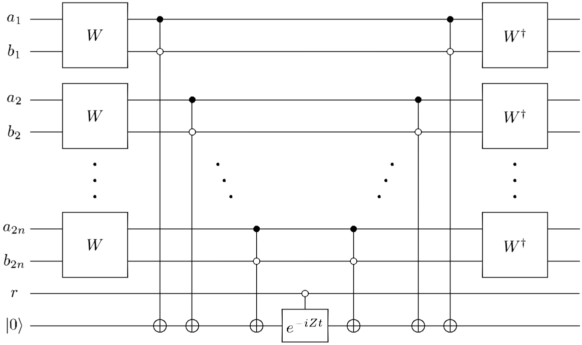
The algorithms create circuits whose sizes and shapes depend on the parameters. E.g. the size of the input register:



(QFT)

Case study: circuit snippets

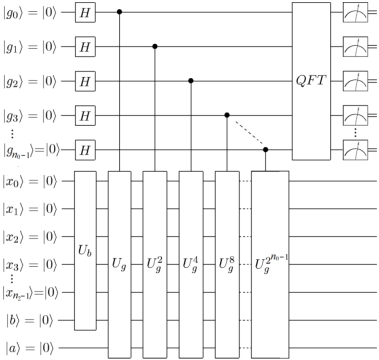
The algorithms create circuits whose sizes and shapes depend on the parameters. E.g. the size of the input register:



(diffusion step in BWT)

Case study: circuit snippets

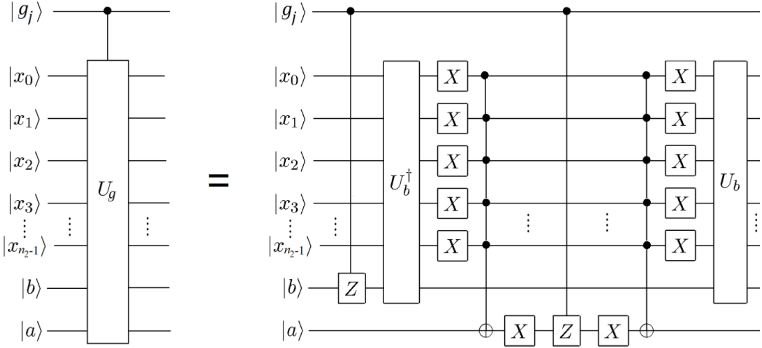
The algorithms create circuits whose sizes and shapes depend on the parameters. E.g. the size of the input register:



(piece of one subroutine of QLS)

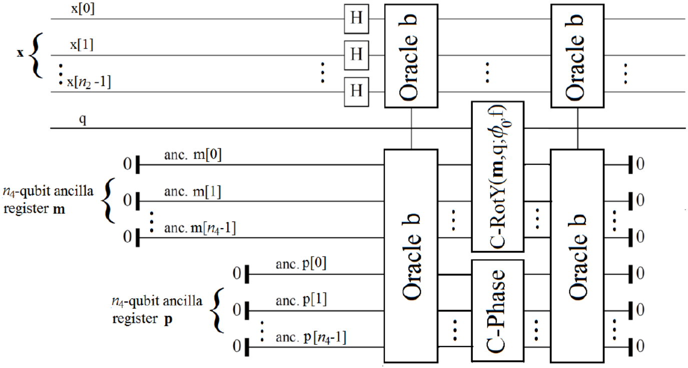
Case study: circuit snippets

The algorithms create circuits whose sizes and shapes depend on the parameters. E.g. the size of the input register:



(the subroutine U_g)

The algorithms create circuits whose sizes and shapes depend on the parameters. E.g. the size of the input register:



(the subroutine U_b)

Plan

Lessons learned

- Circuit construction
  - Procedural: Instruction-based, one line at a time
  - Declarative: Circuit combinators
    - Inversion
    - Repetition
    - Control
    - Computation/uncomputation
- Circuits as inputs to other circuits
- Regularity with respect to the size of the input
- Distinction parameter / input
- Need for automation for oracle generation

Plan

Programming framework

Two approaches

- Circuit as a record
  - One type `circuit`
  - Qubits $\equiv$ wire numbers
  - Native: vertical/horizontal concatenation, gate addition
- Circuit as a function
  - Qubits $\equiv$ first-order objects
  - Input wires $\equiv$ function input
  - Output wires $\equiv$ function output

Circuits as Records

Simplest model: an object holding all of the circuit structure

- Classical wires
- Quantum wires
- List of gates (or directed acyclic graph)
- This is for instance QisKit/QASM model

In this system

- Static circuit
- No high-level hybrid interaction: sequence
  1. circuit generation
  2. circuit evaluation
  3. measure
  4. classical post-processing
  5. back to (1)

Circuits as Records

Procedural construction (QisKit)

```
q = QuantumRegister(5)
c = ClassicalRegister(1)
circ = QuantumCircuit(q,c)

circ.h(q[0])
for i in range(1,5):
  circ.cx(q[0], q[i])
circ.meas(q[4],c[0])
```

- Static ID For registers
- Wires are numbers
- Gate $\equiv$ instruction
- Classical control: Circuit building
- Explicit "run" of circuit

Combinators: return a record `circuit`

- `circ.control(4)`
- `circ.inverse()`
- `circ.append(other-circuit)`

Circuits as Functions

A function (Quipper)

```
a -> Circ b
```

- Inputs something of type `a`
- Outputs something of type `b`
- As a side-effect, generates a circuit snippet.

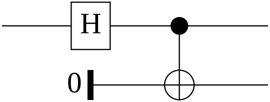
Or

- Inputs a value of type `a`
- Outputs a computation of type `b`

Plan

Circuits as Functions

The circuit



can be typed with

```
Qubit -> Circ (Qubit,Qubit)
```

- Inputs one qubit
- Outputs a pair of qubits
- Spits out some gates when evaluated

The gates are however encapsulated in the function

Circuits as Functions

Representing circuits (Quipper)

Name of circuit
Input: one wire
Indeed a circuit
Two output wires

```
myCircuit :: Qubit -> Circ (Qubit, Qubit)

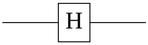
myCircuit q = do
      ...
      ...
      return (x,y)
```

Start a procedural sequence
The name of the input wire
The two output wires

Circuits as Functions

Procedural presentation of circuits:

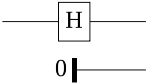
```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Circuits as Functions

Procedural presentation of circuits:

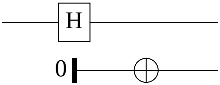
```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Circuits as Functions

Procedural presentation of circuits:

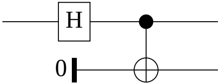
```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Circuits as Functions

Procedural presentation of circuits:

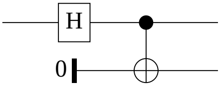
```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



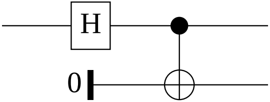
Circuits as Functions

Procedural presentation of circuits:

```
prog :: Qubit -> Circ (Qubit,Qubit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r ‘controlled‘ q
  return (q,r)
```



Circuits as Functions




```
import Quipper

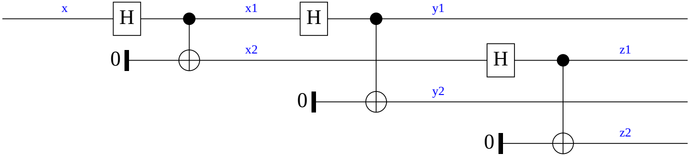
circ ::
   Qubit -> Circ (Qubit,Qubit)
circ x = do
   y <- qinit False
   hadamard_at x
   qnot_at y ‘controlled‘ x
   return (x,y)
```

- Qubits $\equiv$ first-class variable
- Circuit $\equiv$ function
- Wires $\equiv$ inputs and outputs
- Mix classical/quantum

Circuits as Functions

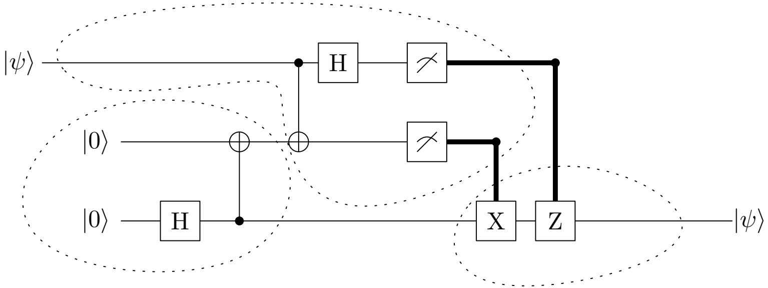
Wires do not have "fixed" location

```
circ2 :: Qubit -> Circ ()
circ2 x = do
   (x1,x2) <- circ x
   (y1,y2) <- circ x1
   (z1,z2) <- circ x2
   return ()
```



- Qubit $\not\equiv$ Wire number
- Circuits as functions: can be applied
- More expressive types

Exercise: Decompose according to the dashed sections



Circuit Combinators: exercise !

What could be the corresponding operations ?

1. `(a -> Circ b) -> (b -> Circ c) -> (a -> Circ c)`
2. `(a -> Circ b) -> (b -> Circ a)`
3. ```
   (a -> Circ b) -> (c -> Circ d)
                 -> ((a,c) -> Circ (b,d))
   ```
4. `(a -> Circ b) -> ((a,Qubit) -> Circ (b,Qubit))`
5. `(a -> Circ b) -> (Qubit -> a -> Circ (b,Qubit))`
6. `(a -> Circ b) -> (Qubit -> a -> Circ b)`
   ```